**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи №1 з дисципліни

«Аналіз даних в інформаційних системах»

„**Створення сховища даних** ”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Тарасьонок Дмитро Євгенович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Олійний Ю. О.*

Київ 2023

ЗМІСТ

[1 Мета лабораторної роботи 5](#_Toc129129905)

[2 Завдання 6](#_Toc129129906)

[1. Самостійно обрати не менше 3-х джерел відкритих даних. 6](#_Toc129129907)

[2. Спроектувати модель Stage зони для ETL процесів. 6](#_Toc129129908)

[3. Спроектувати модель основного сховища за типом «зірка» або «сніжинка». 6](#_Toc129129909)

[4. Створити ETL засоби: 6](#_Toc129129910)

[5. Завантажити дані до основного сховища даних. 6](#_Toc129129911)

[3 Постановка задачі 7](#_Toc129129912)

[3.1 Обрати та завантажити дані 7](#_Toc129129913)

[3.2 Опис джерел даних 7](#_Toc129129914)

[4 Розробка моделі сховища даних 12](#_Toc129129915)

[4.1 Модель stage зони 12](#_Toc129129916)

[4.2 Модель основного сховища даних 13](#_Toc129129917)

[5 Алгоритм завантаження даних 15](#_Toc129129918)

[ДОДАТОК А (Завантаження даних) 17](#_Toc129129919)

[А.1. Загальний код 18](#_Toc129129920)

[А.2. get\_unique\_data() – отримання унікальних номерів рейсів та реєстраційних номерів літаків 18](#_Toc129129921)

[А.3. get\_aircraft\_designators() – отримання моделей літаків 18](#_Toc129129922)

[А.4. get\_aircrafts\_info() – отримання інформації щодо літаків із файлу 18](#_Toc129129923)

[А.5. get\_routes\_info() – отримання інформації щодо польотів із файлу 18](#_Toc129129924)

[А.6. filter\_aircrafts() – фільтрація файлів із літаками для обробки помилок 18](#_Toc129129925)

[А.7. filter\_routes() – фільтрація файлів із польотами для обробки помилок 18](#_Toc129129926)

[А.8. routes\_to\_csv() – об’єднання json файлів із польотами в один CSV файл 18](#_Toc129129927)

[А.9. get\_missing\_aircrafts() – отримання недостаючих літаків 18](#_Toc129129928)

[А.10. aircrafts\_to\_csv() – об’єднання json файлів із літаками в один CSV файл 18](#_Toc129129929)

[А.11. get\_airports() – отримання інформації щодо аеропортів 18](#_Toc129129930)

[А.12. get\_countries() – отримання інформації щодо країн 18](#_Toc129129931)

[А.13. get\_regions() – отримання інформації щодо регіонів 18](#_Toc129129932)

[А.14. get\_airlines() – отримання інформації щодо авіакомпаній 18](#_Toc129129933)

[А.15. generate\_dates() – генерація дат 18](#_Toc129129934)

[А.16. generate\_data() – виклик усіх функцій, необхідних для створення CSV файлів 18](#_Toc129129935)

[ДОДАТОК Б (Скрипти для створення stage зони) 18](#_Toc129129936)

[ДОДАТОК В (Скрипти для створення основного сховища даних) 18](#_Toc129129937)

[Додаток Г (ETL скрипти) 18](#_Toc129129938)

[Г.1. Python-скрипт для завантаження даних у stage зону 18](#_Toc129129939)

[Г.2. SQL-скрипт для завантаження даних зі stage зони до основного сховища даних (окрім вимірів літаків та польотів) 18](#_Toc129129940)

[Г.3. Python-скрипт для створення нової таблиці з літаками та додаванням до неї стовпця назви моделі з таблиці моделей 18](#_Toc129129941)

[Г.4. SQL-скрипт для завантаження даних зі стейдж зони до основного сховища даних (для виміру літаків та польотів) 18](#_Toc129129942)

[Г.5. Python-скрипт для повного завантаження даних 18](#_Toc129129943)

# Мета лабораторної роботи

Мета роботи – ознайомитись з підходами до створення сховищ даних

# Завдання

Навчитися створювати процедури завантаження даних до сховища.

## Самостійно обрати не менше 3-х джерел відкритих даних.

## Спроектувати модель Stage зони для ETL процесів.

## Спроектувати модель основного сховища за типом «зірка» або «сніжинка».

## Створити ETL засоби:

* завантажити дані до Stage зони
* створити набір процедур/функцій для перетворення та завантаження даних до основного сховища (або створити засобами програмних ETL засобів). Передбачити можливість завантаження змінених та додаткових даних.

## Завантажити дані до основного сховища даних.

# Постановка задачі

## Обрати та завантажити дані

Для виконання даної лабораторної роботи мною було обрано дані щодо польотів у світі з сервісу FlightRadar24. Через API сервісу було отримано історію польотів, інформацію про літаки за їх реєстраційними номерами. Для коректного представлення даних також було завантажено список аеропортів світу, список континентів, країн, регіонів країн, а також моделей літаків. Потім для зберігання дат було згенеровано список дат, починаючи з 1 січня 1900 року. Такі старі дати необхідні для зберігання дати виготовлення літаків.

## Опис джерел даних

Загалом було завантажено наступні набори даних:

* 1. Інформація про польоти – була отримана в ході багаторічного збору даних
  2. Історія польотів за номером рейсу: <https://api.flightradar24.com/common/v1/flight/list.json?enc=API_KEY&query=ROUTE_NUM&fetchBy=flight&limit=100&page=1&token=API_KEY>
  3. Інформація про літаки за реєстраційним номером: <https://api.flightradar24.com/common/v1/flight/list.json?enc=API_KEY&query=REGISTRATION&fetchBy=reg&limit=1&page=1&filterBy=&token=API_KEY&client=ios_freemium&version=9.2.1>
  4. Список авіакомпаній: <https://raw.githubusercontent.com/jpatokal/openflights/master/data/airlines.dat>
  5. Список аеропортів: <https://davidmegginson.github.io/ourairports-data/airports.csv>
  6. Список країн: <https://davidmegginson.github.io/ourairports-data/countries.csv>
  7. Список регіонів у країнах: <https://davidmegginson.github.io/ourairports-data/regions.csv>
  8. База даних моделей літаків ICAO: <https://www.icao.int/publications/doc8643/pages/search.aspx>
  9. База даних спеціальних позначень типів літаків ICAO: <https://www.icao.int/publications/DOC8643/Pages/SpecialDesignators.aspx>

Завантаження файлів є автоматизованим і виконується за допомогою функцій мовою Python. Також функції завантаження проводять певну обробку даних. Було сформовано наступні файли:

* 1. aircrafts\_filtered.csv – літаки
  2. airlines.csv – авіакомпанії
  3. airports.csv – аеропорти
  4. continents.csv – континенти
  5. countries.csv – країни
  6. dates.csv – дати
  7. designators.csv – моделі літаків
  8. flights.csv – польоти
  9. regions.csv – регіони
  10. special\_designators – спеціальні позначення типів літаків, конкатенується з файлом designators.csv

Нижче наведена таблиця полів вищезазначених файлів:

|  |  |  |
| --- | --- | --- |
| aircrafts\_filtered.csv | registration | Реєстраційний номер літака |
| model\_code | Код моделі |
| model\_text | Назва моделі |
| country\_code | Код країни реєстрації |
| production\_date | Дата виробництва університету |
| owner\_icao | ICAO код авіакомпанії-власника |
| airlines.csv | airline\_id | Ідентифікатор авіакомпанії |
| name | Назва |
| alias | Псевдонім |
| code | ICAO код авіакомпанії |
| callsign | Позивний |
| country | Країна реєстрації |
| active | Активна? |
| airports.csv | code | ICAO код аеропорту |
| type | Тип |
| name | Назва |
| latitude | Широта |
| longitude | Довгота |
| elevation | Висота над рівнем моря (фт) |
| region | Регіон |
| gps\_code | Код GPS |
| continents.csv | code | ISO код континенту |
| name | Назва |
| countries.csv | code | ISO код країни |
| name | Назва |
| continent | ISO код континенту |
| dates.csv | the\_date | Дата |
| weekday | День тижня (Понеділок – 0) |
| month | Місяць |
| year | Рік |
| quarter | Квартал |
| day\_of\_year | День року (0-365) |
| weekend | Вихідний? |
| week\_of\_year | Номер тижня |
| designators.csv | name | Назва моделі |
| description | Опис |
| turbulence\_category | Категорія турбулентності |
| designator | Позначення |
| manufacturer | Виробник |
| type | Тип |
| engine\_count | Кількість двигунів |
| engine\_type | Тип двигунів |
| flights.csv | flight\_id | Ідентифікатор польоту |
| route\_number | Номер рейсу |
| aircraft\_registration | Реєстраційний номер літака |
| airline\_icao | ICAO код авіакомпанії |
| airport\_origin | Аеропорт вильоту |
| airport\_destination | Аеропорт призначення |
| scheduled\_departure | Запланований час вильоту |
| scheduled\_arrival | Запланований час прильоту |
| real\_departure | Фактичний час вильоту |
| real\_arrival | Фактичний час прильоту |
| regions.csv | code | ISO код регіону |
| name | Назва |
| country | ISO код країни |
| special\_designators | ModelFullName | Назва моделі |
| Designator | Позначення |

# Розробка моделі сховища даних

Сховище даних вміщатиме всю вищезазначену інформацію. Фактова таблиця міститиме інформацію про номер рейсу, авіакомпанію, літак, заплановані та реальні часи вильоту та прильоту.

## Модель stage зони

У stage зоні будуть наступні таблиці:

1. aircrafts - літаки
2. airlines - авіакомпанії
3. airports - аеропорти
4. flights - польоти
5. designators – моделі літаків
6. dates - дати
7. regions - регіони
8. countries - країни
9. continents – континенти

Загалом усі поля цієї моделі співпадають з полями CSV файлів, але варто зауважити, що в кожній таблиці первинним ключем є автоінкремент. Зроблено це для того, щоб при додаванні значень у неочищену stage зону не виникало проблем. На рисунку 4.1 наведена схема stage зони

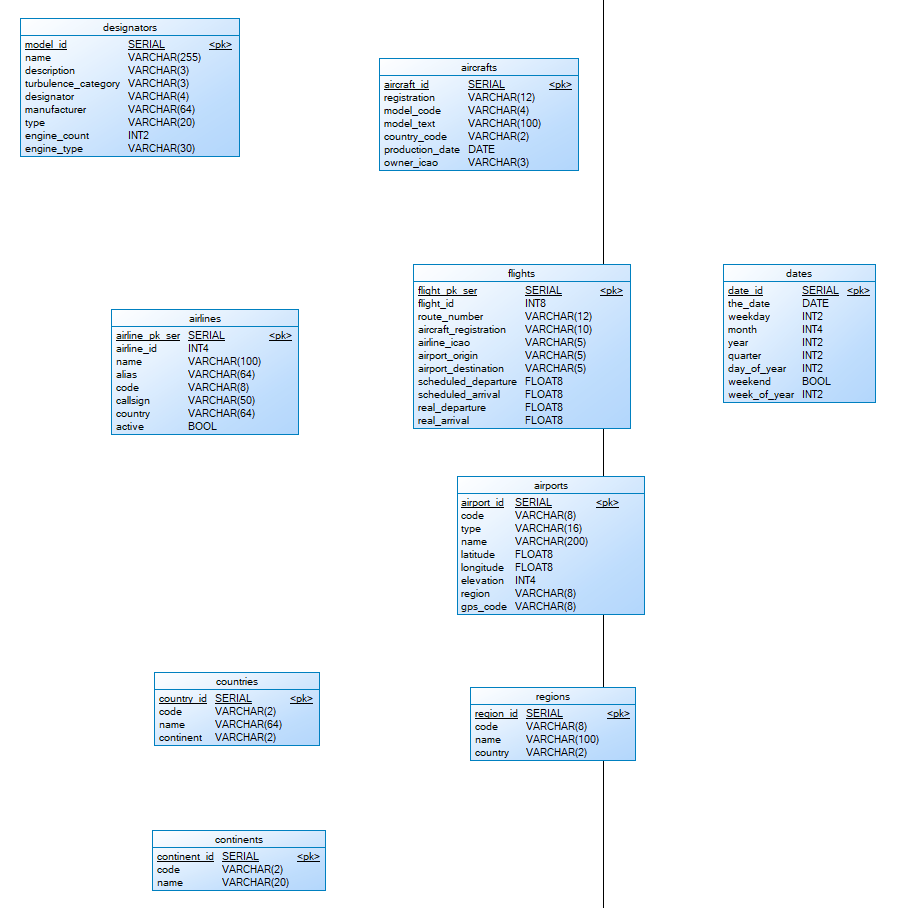


Рисунок 4.1 – Модель stage зони

## Модель основного сховища даних

В основному сховищі всі значення, що можуть повторюватися, були винесені в окремі таблиці. Це: виробники літаків, категорії турбулентності літаків, типи літаків, описи літаків, типи двигунів, маршрути, типи аеропортів.

Загалом основне сховище даних включає такі таблиці:

1. airlines\_dim – вимір авіакомпаній
2. airports\_dim – вимір аеропортів
3. aircraft\_types\_dim – вимір типів літаків
4. countries\_dim – вимір країн
5. airports\_types\_dim – вимір типів аеропортів
6. continents\_dim – вимір континентів
7. engine\_types\_dim – вимір типів двигунів
8. models\_descriptions\_dim – вимір описів моделей літаків
9. manufacturers\_dim – вимір виробників літаків
10. flights\_fact – вимір польотів
11. dates\_dim – вимір дат
12. aircrafts\_dim – вимір літаків
13. designators\_dim – вимір моделей літаків
14. regions\_dim – вимір регіонів
15. turbulence\_categories\_dim – вимір категорій турбулентності
16. routes\_dim – вимір маршрутів

На рисунку 4.2 наведено модель основного сховища даних

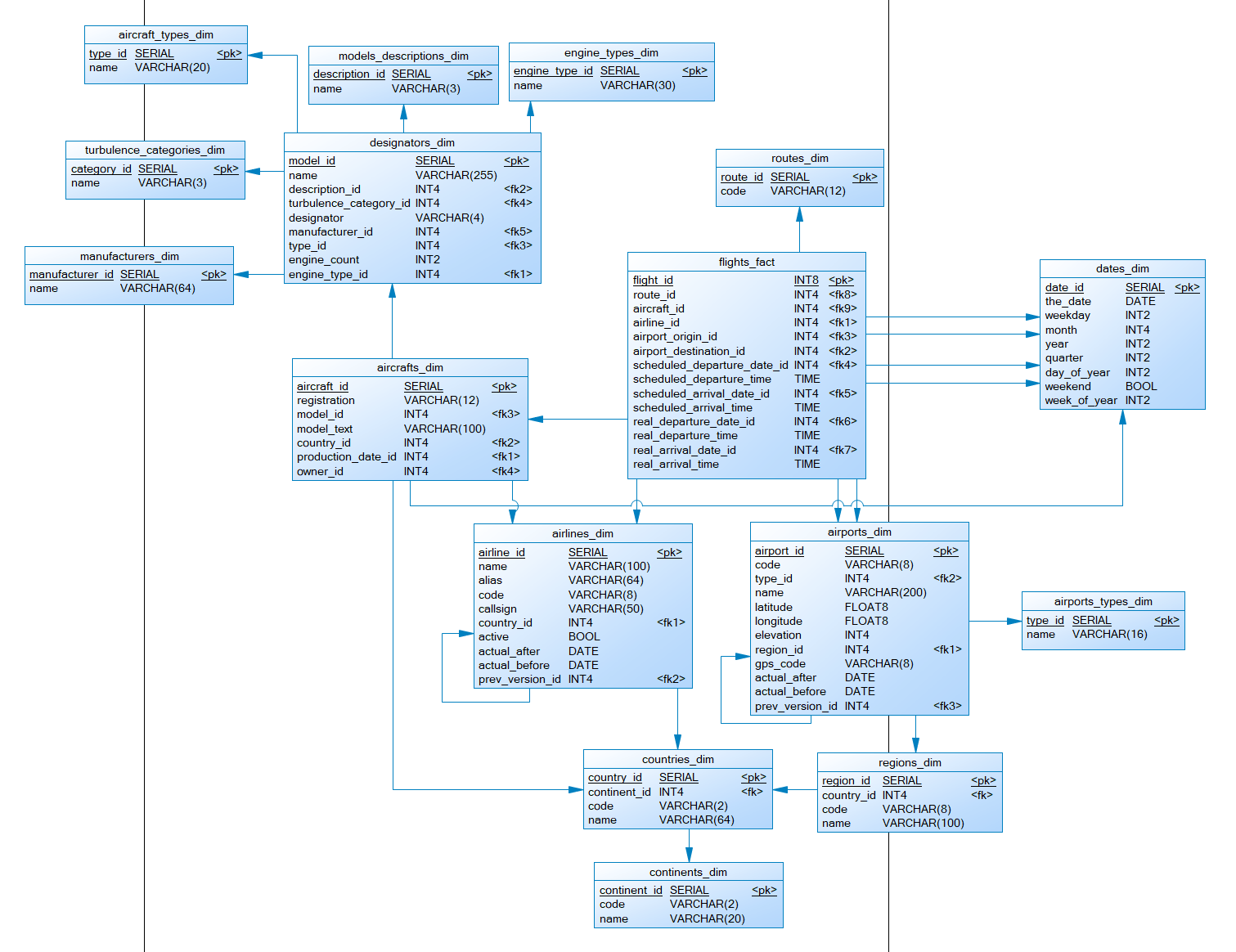


Рисунок 4.2 – Модель основного сховища даних

# Алгоритм завантаження даних

Для початкового завантаження даних мною було створено такий алгоритм:

1. Переглядається історія польотів, отримуються номери рейсів
2. За номерами рейсів завантажуємо інформацію про них у json форматі
3. Об’єднуємо файли json в один файл CSV
4. Переглядаємо отриманий файл CSV та отримуємо унікальні реєстраційні номери літаків
5. Завантажуємо інформацію про літаки у json форматі
6. Об’єднуємо файли json в один файл CSV
7. Завантажуємо іншу зазначену інформацію
8. Генеруємо дати

Варто вказати, що цей алгоритм включає відкидання зайвих стовпців у файлах даних, а тому вихідні CSV файли вже мають такий же вигляд, як і таблиці stage зони, через що надалі з ними не потрібно проводити ніяких маніпуляцій.

Після створення CSV файлів дані завантажуються у stage зону, яка перед цим повністю очищається, але у випадку, якщо цього зроблено не було, зона спроектована таким чином, щоб це не викликало проблем.

Коли дані завантажено в stage зону, є декілька варіантів подальших дій з перенесення даних зі stage зони до основного сховища

1. Якщо таблиця виміру не містить зовнішніх ключів, треба завантажити всі унікальні значення цієї таблиці в stage зоні до таблиці вимірів при цьому ігноруючи повтори значень
2. Якщо таблиця виміру містить зовнішні ключі, треба отримати всі значення за унікальним полем, відсортовані за спаданням індексу (для того щоб додавалися новіші значення), а потім об’єднати за допомогою LEFT JOIN там, де значення може бути нульовим та INNER JOIN, де не може
3. Якщо це таблиця aircrafts, необхідно викликати функцію на мові Python, яка знайде назву моделі з таблиці моделей, а потім об’єднати ці таблиці

В останніх двох випадках при завантаженні одних і тих самих даних треба перевіряти цей момент і оновлювати значення для цього поля (ON CONFLICT DO UPDATE)

# ДОДАТОК А (Завантаження даних)

У даному додатку наведено код на мові Python для завантаження даних. Варто зауважити, що автоматичне завантаження файлів continents.csv, special\_designators.csv не передбачене.

## Загальний код

import csv  
import json  
import re  
import time  
import uuid  
from pathlib import Path  
  
import numpy as np  
import pandas as pd  
import pytz  
import requests  
import datetime as dt  
import tqdm  
import psycopg2  
from difflib import SequenceMatcher  
*#%%*DATA\_DIR = Path.cwd().joinpath('data')  
SQL\_DIR = Path.cwd().joinpath('sql')  
AIRCRAFTS\_DIR = DATA\_DIR.joinpath('aircrafts')  
AIRCRAFTS\_FILTERED\_DIR = DATA\_DIR.joinpath('aircrafts\_filtered')  
ROUTES\_DIR = DATA\_DIR.joinpath('routes')  
ROUTES\_FILTERED\_DIR = DATA\_DIR.joinpath('routes\_filtered')  
*#%%*DB\_HOST = 'localhost'  
DB\_PORT = 5432  
DB\_USER = 'postgres'  
DB\_PASSWORD = r'zWKHqx1N3%Gt'  
DB\_NAME = 'data\_analysis\_lab1'  
*#%%*FR24\_DTYPES = {  
 'flight\_id': object,  
 'icao24': object,  
 'latitude': float,  
 'longitude': float,  
 'heading': int,  
 'height': int,  
 'airspeed': int,  
 'squawk': float,  
 'locator': object,  
 'aircraft': object,  
 'registration': object,  
 'unixtime': int,  
 'departure': object,  
 'arrival': object,  
 'ticket\_route': object,  
 'status': int,  
 'vertical\_speed': int,  
 'transponder\_route': object,  
 'airline': object,  
}  
*#%%*SYMBOL\_REPLACES = {  
 '\u00c0': 'A',  
 '\u00c1': 'A',  
 '\u00c2': 'A',  
 '\u00c3': 'A',  
 '\u00c4': 'A',  
 '\u00c5': 'A',  
 '\u00c6': 'A',  
 '\u00c7': 'C',  
 '\u00c8': 'E',  
 '\u00c9': 'E',  
 '\u00ca': 'E',  
 '\u00cb': 'E',  
 '\u00cc': 'I',  
 '\u00cd': 'I',  
 '\u00ce': 'I',  
 '\u00cf': 'I',  
 '\u00d1': 'N',  
 '\u00d2': 'O',  
 '\u00d3': 'O',  
 '\u00d4': 'O',  
 '\u00d5': 'O',  
 '\u00d6': 'O',  
 '\u00d8': 'O',  
 '\u00d9': 'U',  
 '\u00da': 'U',  
 '\u00db': 'U',  
 '\u00dc': 'U',  
 '\u00dd': 'Y',  
 '\u00df': 'S',  
 '\u00e0': 'a',  
 '\u00e1': 'a',  
 '\u00e2': 'a',  
 '\u00e3': 'a',  
 '\u00e4': 'a',  
 '\u00e5': 'a',  
 '\u00e6': 'a',  
 '\u00e7': 'c',  
 '\u00e8': 'e',  
 '\u00e9': 'e',  
 '\u00ea': 'e',  
 '\u00eb': 'e',  
 '\u00ec': 'i',  
 '\u00ed': 'i',  
 '\u00ee': 'i',  
 '\u00ef': 'i',  
 '\u00f0': 'd',  
 '\u00f1': 'n',  
 '\u00f2': 'o',  
 '\u00f3': 'o',  
 '\u00f4': 'o',  
 '\u00f5': 'o',  
 '\u00f6': 'ö',  
 '\u00f8': 'o',  
 '\u00f9': 'u',  
 '\u00fa': 'u',  
 '\u00fb': 'u',  
 '\u00fc': 'u',  
 '\u00fd': 'y',  
 '\u00ff': 'y',  
 '\u200b': '',  
 '\xa0': ' '  
}

## get\_unique\_data() – отримання унікальних номерів рейсів та реєстраційних номерів літаків

def get\_unique\_data():  
 registrations = []  
 routes = []  
 for index, file in enumerate(  
 list(DATA\_DIR.joinpath('fr24').iterdir())[0:]):  
 try:  
 data = pd.read\_csv(file, names=FR24\_DTYPES.keys(), on\_bad\_lines='skip', dtype=FR24\_DTYPES)  
 except ValueError:  
 continue  
 unique\_registrations = data.registration.dropna().unique()  
 registrations.append(unique\_registrations)  
  
 unique\_routes = data.ticket\_route.dropna().unique()  
 routes.append(unique\_routes)  
 registrations = np.unique(np.concatenate(registrations))  
 routes = np.unique(np.concatenate(routes))  
 return registrations, routes

## get\_aircraft\_designators() – отримання моделей літаків

def get\_aircraft\_designators():  
 response = requests.post(  
 url='https://www4.icao.int/doc8643/External/AircraftTypes'  
 )  
  
 content = response.content.decode(response.encoding)  
 json\_content = json.loads(content)  
 designators\_data = pd.DataFrame.from\_records(json\_content)  
 special\_designators = pd.read\_csv(DATA\_DIR.joinpath('special\_designators.csv'))  
  
 designators = pd.concat([designators\_data, special\_designators])  
  
 designators.columns = ['name', 'description', 'turbulence\_category', 'WTG', 'designator', 'manufacturer', 'type',  
 'engine\_count', 'engine\_type']  
 designators.engine\_count = designators.engine\_count.str.replace(r'[^\d]+', '1', regex=True)  
  
 designators.to\_csv(DATA\_DIR.joinpath('designators.csv'), index=False,  
 columns=['name', 'description', 'turbulence\_category', 'designator', 'manufacturer', 'type',  
 'engine\_count', 'engine\_type'])

## get\_aircrafts\_info() – отримання інформації щодо літаків із файлу

def get\_aircrafts\_info(filename):  
 registrations = pd.read\_csv(filename, names=['registration'])  
  
 registrations = registrations.sample(frac=1)  
  
 progress\_bar = tqdm.tqdm(registrations.registration)  
 for registration in progress\_bar:  
 progress\_bar.set\_description(f'Fetching {registration}')  
 try:  
 response = requests.get(  
 url='https://api.flightradar24.com/common/v1/flight/list.json?'  
 'enc=IKQGxn3NR31\_n-55iS2uKcuzjmvSFrtJX6mpRJYT7oI&'  
 f'query={registration}&'  
 'fetchBy=reg&'  
 'limit=1&'  
 'timestamp=0&'  
 'page=1&'  
 'filterBy=&'  
 'token=IKQGxn3NR31\_n-55iS2uKcuzjmvSFrtJX6mpRJYT7oI&'  
 'client=ios\_freemium&'  
 'version=9.2.1',  
 headers={  
 'User-Agent': 'FlightradarFree/2023021501 CFNetwork/1404.0.5 Darwin/22.3.0'  
 }  
 )  
 with open(f'aircrafts\\{registration}.json', 'w', encoding=response.encoding) as file:  
 file.write(response.content.decode(response.encoding))  
 time.sleep(0.6)  
 except:  
 ...

## get\_routes\_info() – отримання інформації щодо польотів із файлу

def get\_routes\_info(filename: 'str | Path'):  
 routes = pd.read\_csv(filename, names=['route'])  
  
 routes = routes.sample(frac=1)  
  
 progress\_bar = tqdm.tqdm(routes.route)  
  
 for route in progress\_bar:  
 progress\_bar.set\_description(f'Fetching {routes}')  
 progress\_bar.refresh()  
 page = 1  
 last = None  
 while True:  
 time.sleep(0.6)  
  
 response = requests.get(  
 url='https://api.flightradar24.com/common/v1/flight/list.json?'  
 'enc=6pDfb2KZPxots\_3kFVasmNL1WJ7rQXvJ5yJb4NhegjA&'  
 f'query={route}&'  
 'fetchBy=flight&'  
 'limit=100&'  
 f'page={page}&'  
 'token=6pDfb2KZPxots\_3kFVasmNL1WJ7rQXvJ5yJb4NhegjA',  
 headers={  
 'User-Agent': 'FlightradarFree/2023021501 CFNetwork/1404.0.5 Darwin/22.3.0'  
 }  
 )  
 content = response.content.decode(response.encoding)  
  
 debug\_name = DATA\_DIR.joinpath(f'debug\\{uuid.uuid4()}.json')  
  
 with open(debug\_name, 'w') as file:  
 file.write(content)  
  
 if '402 Payment Required' in content or 'Error reference number' in content:  
 break  
  
 json\_data = json.loads(content)  
 if 'errors' in json\_data:  
 break  
 elif 'result' not in json\_data:  
 break  
 elif 'response' not in json\_data['result']:  
 break  
  
 result = json\_data['result']['response']  
  
 respath = ROUTES\_DIR.joinpath(f'{route}\_{page}.json')  
  
 with open(respath, 'w') as file:  
 json.dump(json\_data, file)  
  
 if 'page' not in result:  
 break  
  
 data = result['data']  
  
 if data is None or (last is not None and data[0] == last):  
 respath.unlink()  
 break  
  
 if not result['page']['more']:  
 break  
  
 page += 1  
 last = data[0]

## filter\_aircrafts() – фільтрація файлів із літаками для обробки помилок

def filter\_aircrafts():  
 progress\_bar = tqdm.tqdm(AIRCRAFTS\_DIR.iterdir())  
 for filename in progress\_bar:  
 progress\_bar.set\_description(f'Checking {filename.name}')  
 if filename.suffix == '.json':  
 with open(filename) as file:  
 content = file.read()  
 if '402 Payment Required' in content or 'Cloudflare Location' in content:  
 filename.unlink()  
 else:  
 json\_data = json.loads(content)  
 if 'errors' not in json\_data:  
 aircraftInfo = json\_data['result']['response']['aircraftInfo']  
 if aircraftInfo:  
 with open(DATA\_DIR.joinpath('aircrafts\_filtered').joinpath(filename.name), 'w') as file:  
 json.dump(json\_data['result']['response']['aircraftInfo'], file)  
 else:  
 filename.unlink()

## filter\_routes() – фільтрація файлів із польотами для обробки помилок

def filter\_routes():  
 progress\_bar = tqdm.tqdm(list(ROUTES\_DIR.iterdir()))  
 for filename in progress\_bar:  
 progress\_bar.set\_description(f'Checking {filename.name}')  
 if filename.suffix == '.json':  
 with open(filename) as file:  
 content = file.read()  
 json\_data = json.loads(content)  
 if 'errors' not in json\_data:  
 response = json\_data['result']['response']  
 if 'data' in response and response['data']:  
 with open(ROUTES\_FILTERED\_DIR.joinpath(filename.name), 'w') as file:  
 json.dump(response['data'], file)

## routes\_to\_csv() – об’єднання json файлів із польотами в один CSV файл

def routes\_to\_csv():  
 def check\_data(data) -> bool:  
 if 'identification' in data:  
 if 'id' in data['identification']:  
 return data['identification']['id'] is not None  
 return False  
  
 csv\_path = DATA\_DIR.joinpath('flights.csv')  
 if csv\_path.exists():  
 csv\_path.unlink()  
  
 progress\_bar = tqdm.tqdm(list(ROUTES\_FILTERED\_DIR.iterdir()))  
  
 for filename in progress\_bar:  
 progress\_bar.set\_description(f'Adding {filename.name}')  
 progress\_bar.refresh()  
 with open(filename) as json\_file:  
 json\_data = json.load(json\_file)  
 airline\_icao = ''  
 flights = []  
 for index, row in enumerate(filter(check\_data, json\_data)):  
 if 'status' in row:  
 if row['status']['live']:  
 continue  
 route\_number\_regex = re.match('(?P<number>.+)\_.+', filename.name)  
 route\_number = route\_number\_regex.group('number')  
 flight\_id = row['identification']['id']  
 aircraft\_registration = ''  
 airport\_origin = ''  
 airport\_destination = ''  
 scheduled\_departure = ''  
 scheduled\_arrival = ''  
 real\_departure = ''  
 real\_arrival = ''  
  
 if 'aircraft' in row:  
 aircraft = row['aircraft']  
 if aircraft and 'registration' in aircraft:  
 aircraft\_registration = row['aircraft']['registration']  
 if not airline\_icao and 'airline' in row:  
 airline = row['airline']  
 if airline and 'code' in airline:  
 code = airline['code']  
 if code and 'icao' in code:  
 airline\_icao = code['icao']  
 if 'airport' in row:  
 airport = row['airport']  
 if airport:  
 if 'origin' in airport:  
 origin = airport['origin']  
 if origin and 'code' in origin:  
 code = origin['code']  
 if code and 'icao' in code:  
 airport\_origin = code['icao']  
 if 'destination' in airport:  
 destination = airport['destination']  
 if destination and 'code' in destination:  
 code = destination['code']  
 if code and 'icao' in code:  
 airport\_destination = code['icao']  
 if 'time' in row:  
 route\_times = row['time']  
 if route\_times:  
 if 'scheduled' in route\_times:  
 scheduled\_times = route\_times['scheduled']  
 if scheduled\_times:  
 if 'departure' in scheduled\_times:  
 scheduled\_departure = scheduled\_times['departure']  
 if 'arrival' in scheduled\_times:  
 scheduled\_arrival = scheduled\_times['arrival']  
 if 'real' in route\_times:  
 real\_times = route\_times['real']  
 if real\_times:  
 if 'departure' in real\_times:  
 real\_departure = real\_times['departure']  
 if 'arrival' in real\_times:  
 real\_arrival = real\_times['arrival']  
 flights.append([int(flight\_id, 16), route\_number, aircraft\_registration, airline\_icao,  
 airport\_origin, airport\_destination,  
 scheduled\_departure, scheduled\_arrival, real\_departure,  
 real\_arrival])  
 if airline\_icao:  
 for flight in flights:  
 flight[3] = airline\_icao  
 with open(csv\_path, 'a', newline='') as file:  
 csv\_writer = csv.writer(file)  
 csv\_writer.writerows(flights)  
  
 dtypes = {  
 'flight\_id': int,  
 'route\_number': object,  
 'aircraft\_registration': object,  
 'airline\_icao': object,  
 'airport\_origin': object,  
 'airport\_destination': object,  
 'scheduled\_departure': object,  
 'scheduled\_arrival': object,  
 'real\_departure': float,  
 'real\_arrival': float  
 }  
 data = pd.read\_csv(  
 csv\_path,  
 names=dtypes.keys()  
 )  
  
 data = data.drop\_duplicates(subset='flight\_id')  
 data.to\_csv(csv\_path, index=False)

## get\_missing\_aircrafts() – отримання недостаючих літаків

def get\_missing\_aircrafts():  
 dtypes = {  
 'flight\_id': str,  
 'route\_number': str,  
 'aircraft\_registration': str,  
 'airline\_icao': str,  
 'airport\_origin': str,  
 'airport\_destination': str,  
 'scheduled\_departure': str,  
 'scheduled\_arrival': str,  
 'real\_departure': float,  
 'real\_arrival': float  
 }  
 data = pd.read\_csv(  
 DATA\_DIR.joinpath('routes\_filtered.csv'),  
 names=dtypes.keys(),  
 dtype=dtypes,  
 index\_col='flight\_id'  
 )  
  
 aircrafts\_aircrafts = set(map(lambda p: p.stem, DATA\_DIR.joinpath('aircrafts\_filtered').iterdir()))  
 routes\_aircrafts = set(data.aircraft\_registration.unique())  
  
 with open(DATA\_DIR.joinpath('missed\_aircrafts.csv'), 'w') as file:  
 file.write('\n'.join(map(str, routes\_aircrafts - aircrafts\_aircrafts)))  
  
 get\_aircrafts\_info(DATA\_DIR.joinpath('missed\_aircrafts.csv'))

## aircrafts\_to\_csv() – об’єднання json файлів із літаками в один CSV файл

def aircrafts\_to\_csv():  
 csv\_path = DATA\_DIR.joinpath('aircrafts\_filtered.csv')  
 with open(csv\_path, 'w', newline='') as csv\_file:  
 csw\_writer = csv.writer(csv\_file)  
 csw\_writer.writerow([  
 'registration',  
 'model\_code',  
 'model\_text',  
 'country\_code',  
 'production\_date',  
 'owner\_icao'  
 ])  
  
 progress\_bar = tqdm.tqdm(list(AIRCRAFTS\_FILTERED\_DIR.iterdir()))  
  
 for filename in progress\_bar:  
 *# for filename in [AIRCRAFTS\_FILTERED\_DIR.joinpath('D-EEEH.json')]:* progress\_bar.set\_description(f'Adding {filename.name}')  
 progress\_bar.refresh()  
 with open(filename) as json\_file:  
 try:  
 json\_data = json.load(json\_file)  
 except:  
 continue  
 registration = filename.stem  
  
 model\_code = None  
 model\_text = None  
 country\_code = None  
 production\_date = None  
 owner\_icao = None  
  
 if 'model' in json\_data:  
 model = json\_data['model']  
 if model:  
 if 'code' in model:  
 model\_code = model['code']  
 if 'text' in model:  
 model\_text = model['text']  
 if model\_text:  
 for original, to\_replace in SYMBOL\_REPLACES.items():  
 model\_text = model\_text.replace(original, to\_replace)  
  
 if 'country' in json\_data:  
 country = json\_data['country']  
 if country and 'alpha2' in country:  
 country\_code = country['alpha2']  
  
 if 'age' in json\_data:  
 age = json\_data['age']  
 if age and 'date' in age and age['date']:  
 production\_date = dt.datetime.strptime(age['date'], '%b %Y').date()  
  
 if 'owner' in json\_data:  
 owner = json\_data['owner']  
 if owner and 'code' in owner:  
 code = owner['code']  
 if 'icao' in code:  
 owner\_icao = code['icao']  
 with open(csv\_path, 'a', newline='') as file:  
 csv\_writer = csv.writer(file)  
 csv\_writer.writerow([  
 registration,  
 model\_code,  
 model\_text,  
 country\_code,  
 production\_date,  
 owner\_icao  
 ])

## get\_airports() – отримання інформації щодо аеропортів

def get\_airports():  
 airports = pd.read\_csv('https://davidmegginson.github.io/ourairports-data/airports.csv', header=0, names=[  
 'airport\_id',  
 'code',  
 'type',  
 'name',  
 'latitude',  
 'longitude',  
 'elevation',  
 'continent',  
 'country',  
 'region',  
 'municipality',  
 'scheduled\_service',  
 'gps\_code',  
 'iata\_code',  
 'local\_code',  
 'home\_link',  
 'wikipedia\_link',  
 'keywords'  
 ], keep\_default\_na=False)  
 airports.to\_csv(DATA\_DIR.joinpath('airports.csv'), columns=[  
 'code',  
 'type',  
 'name',  
 'latitude',  
 'longitude',  
 'elevation',  
 'region',  
 'gps\_code'  
 ], index=False)

## get\_countries() – отримання інформації щодо країн

def get\_countries():  
 countries = pd.read\_csv('https://davidmegginson.github.io/ourairports-data/countries.csv', header=0, names=[  
 'country\_id',  
 'code',  
 'name',  
 'continent',  
 'wikipedia\_link',  
 'keywords'  
 ], keep\_default\_na=False)  
 countries.to\_csv(DATA\_DIR.joinpath('countries.csv'), columns=[  
 'code',  
 'name',  
 'continent'  
 ], index=False)

## get\_regions() – отримання інформації щодо регіонів

def get\_regions():  
 regions = pd.read\_csv('https://davidmegginson.github.io/ourairports-data/regions.csv', header=0, names=[  
 'region\_id',  
 'code',  
 'local\_coe',  
 'name',  
 'continent',  
 'country',  
 'wikipedia\_link',  
 'keywords'  
 ], keep\_default\_na=False)  
 regions.to\_csv(DATA\_DIR.joinpath('regions.csv'), columns=[  
 'code',  
 'name',  
 'country'  
 ], index=False)

## get\_airlines() – отримання інформації щодо авіакомпаній

def get\_airlines():  
 airlines = pd.read\_csv('https://raw.githubusercontent.com/jpatokal/openflights/master/data/airlines.dat',  
 header=None, names=[  
 'airline\_id',  
 'name',  
 'alias',  
 'iata\_code',  
 'code',  
 'callsign',  
 'country',  
 'active'  
 ], keep\_default\_na=False)  
 airlines.active = airlines.active.str.upper().map({'Y': True, 'N': False})  
 airlines.to\_csv(DATA\_DIR.joinpath('airlines.csv'), columns=[  
 'airline\_id',  
 'name',  
 'alias',  
 'code',  
 'callsign',  
 'country',  
 'active'  
 ], index=False)

## generate\_dates() – генерація дат

def generate\_dates(  
 start\_date=dt.datetime(year=1900, month=1, day=1, tzinfo=pytz.UTC),  
 last\_date=dt.datetime.now(tz=pytz.UTC)  
):  
 if start\_date.tzinfo != pytz.UTC or last\_date.tzinfo != pytz.UTC:  
 raise ValueError('Time zone must be UTC')  
  
 rows = [  
 ['the\_date', 'weekday', 'month', 'year', 'quarter', 'day\_of\_year', 'weekend', 'week\_of\_year']  
 ]  
  
 first\_unix\_date = dt.datetime(year=1970, month=1, day=1, tzinfo=pytz.UTC)  
 current\_date = start\_date  
  
 while current\_date < last\_date:  
 the\_date = current\_date.date()  
 weekday = current\_date.weekday()  
 month = current\_date.month  
 year = current\_date.year  
 quarter = (current\_date.month - 1) // 3  
 day\_of\_year = current\_date.timetuple().tm\_yday  
 weekend = 5 <= weekday  
 week\_of\_year = current\_date.isocalendar()[1]  
 rows.append([the\_date, weekday, month, year, quarter, day\_of\_year, weekend, week\_of\_year])  
  
 with open(DATA\_DIR.joinpath('dates.csv'), 'w', newline='') as csv\_file:  
 csw\_writer = csv.writer(csv\_file)  
 csw\_writer.writerows(rows)

## generate\_data() – виклик усіх функцій, необхідних для створення CSV файлів

def generate\_data():  
 aircrafts\_to\_csv()  
 get\_airlines()  
 get\_airports()  
 get\_countries()  
 generate\_dates()  
 get\_aircraft\_designators()  
 routes\_to\_csv()  
 get\_regions()

# ДОДАТОК Б (Скрипти для створення stage зони)

*/\*==============================================================\*/  
/\* DBMS name: PostgreSQL 9.x \*/  
/\* Created on: 07.03.2023 0:44:41 \*/  
/\*==============================================================\*/*drop table if exists stage.aircrafts;  
  
drop table if exists stage.airlines;  
  
drop table if exists stage.airports;  
  
drop table if exists stage.continents;  
  
drop table if exists stage.countries;  
  
drop table if exists stage.dates;  
  
drop table if exists stage.designators;  
  
drop table if exists stage.flights;  
  
drop table if exists stage.regions;  
  
*/\*==============================================================\*/  
/\* Table: aircrafts \*/  
/\*==============================================================\*/*create table stage.aircrafts  
(  
 aircraft\_id SERIAL not null,  
 registration VARCHAR(12) not null,  
 model\_code VARCHAR(4) null,  
 model\_text VARCHAR(100) null,  
 country\_code VARCHAR(2) null,  
 production\_date DATE null,  
 owner\_icao VARCHAR(3) null,  
 constraint PK\_AIRCRAFTS primary key (aircraft\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: airlines \*/  
/\*==============================================================\*/*create table stage.airlines  
(  
 airline\_pk\_ser SERIAL not null,  
 airline\_id INT4 not null,  
 name VARCHAR(100) not null,  
 alias VARCHAR(64) null,  
 code VARCHAR(8) null,  
 callsign VARCHAR(50) null,  
 country VARCHAR(64) null,  
 active BOOL not null,  
 constraint PK\_AIRLINES primary key (airline\_pk\_ser)  
);  
  
*/\*==============================================================\*/  
/\* Table: airports \*/  
/\*==============================================================\*/*create table stage.airports  
(  
 airport\_id SERIAL not null,  
 code VARCHAR(8) null,  
 type VARCHAR(16) not null,  
 name VARCHAR(200) not null,  
 latitude FLOAT8 not null,  
 longitude FLOAT8 not null,  
 elevation INT4 null,  
 region VARCHAR(8) not null,  
 gps\_code VARCHAR(8) null,  
 constraint PK\_AIRPORTS primary key (airport\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: continents \*/  
/\*==============================================================\*/*create table stage.continents  
(  
 continent\_id SERIAL not null,  
 code VARCHAR(2) not null,  
 name VARCHAR(20) not null,  
 constraint PK\_CONTINENTS primary key (continent\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: countries \*/  
/\*==============================================================\*/*create table stage.countries  
(  
 country\_id SERIAL not null,  
 code VARCHAR(2) not null,  
 name VARCHAR(64) not null,  
 continent VARCHAR(2) not null,  
 constraint PK\_COUNTRIES primary key (country\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: dates \*/  
/\*==============================================================\*/*create table stage.dates  
(  
 date\_id SERIAL not null,  
 the\_date DATE not null,  
 weekday INT2 not null,  
 month INT4 not null,  
 year INT2 not null,  
 quarter INT2 not null,  
 day\_of\_year INT2 not null,  
 weekend BOOL not null,  
 week\_of\_year INT2 not null,  
 constraint PK\_DATES primary key (date\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: designators \*/  
/\*==============================================================\*/*create table stage.designators  
(  
 model\_id SERIAL not null,  
 name VARCHAR(255) not null,  
 description VARCHAR(3) null,  
 turbulence\_category VARCHAR(3) null,  
 designator VARCHAR(4) not null,  
 manufacturer VARCHAR(64) null,  
 type VARCHAR(20) null,  
 engine\_count INT2 null,  
 engine\_type VARCHAR(30) null,  
 constraint PK\_DESIGNATORS primary key (model\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: flights \*/  
/\*==============================================================\*/*create table stage.flights  
(  
 flight\_pk\_ser SERIAL not null,  
 flight\_id INT8 not null,  
 route\_number VARCHAR(12) not null,  
 aircraft\_registration VARCHAR(10) null,  
 airline\_icao VARCHAR(5) null,  
 airport\_origin VARCHAR(5) null,  
 airport\_destination VARCHAR(5) null,  
 scheduled\_departure FLOAT8 null,  
 scheduled\_arrival FLOAT8 null,  
 real\_departure FLOAT8 null,  
 real\_arrival FLOAT8 null,  
 constraint PK\_FLIGHTS primary key (flight\_pk\_ser)  
);  
  
*/\*==============================================================\*/  
/\* Table: regions \*/  
/\*==============================================================\*/*create table stage.regions  
(  
 region\_id SERIAL not null,  
 code VARCHAR(8) not null,  
 name VARCHAR(100) not null,  
 country VARCHAR(2) not null,  
 constraint PK\_REGIONS primary key (region\_id)  
);

# ДОДАТОК В (Скрипти для створення основного сховища даних)

*/\*==============================================================\*/  
/\* DBMS name: PostgreSQL 9.x \*/  
/\* Created on: 07.03.2023 0:45:27 \*/  
/\*==============================================================\*/*drop index if exists aircraft\_type\_name\_unique cascade;  
  
drop table if exists aircraft\_types\_dim cascade;  
  
drop index if exists aircraft\_registration\_unique cascade;  
  
drop table if exists aircrafts\_dim cascade;  
  
drop index if exists airline\_code\_unique cascade;  
  
drop table if exists airlines\_dim cascade;  
  
drop index if exists airport\_code\_unique cascade;  
  
drop table if exists airports\_dim cascade;  
  
drop index if exists airport\_type\_name\_unique cascade;  
  
drop table if exists airports\_types\_dim cascade;  
  
drop index if exists continent\_code\_unique cascade;  
  
drop table if exists continents\_dim cascade;  
  
drop index if exists country\_code\_unique cascade;  
  
drop table if exists countries\_dim cascade;  
  
drop index if exists time\_date\_unique cascade;  
  
drop table if exists dates\_dim cascade;  
  
drop index if exists designator\_unique cascade;  
  
drop table if exists designators\_dim cascade;  
  
drop index if exists engine\_type\_name\_unique cascade;  
  
drop table if exists engine\_types\_dim cascade;  
  
drop table if exists flights\_fact cascade;  
  
drop index if exists manufacturer\_name\_unique cascade;  
  
drop table if exists manufacturers\_dim cascade;  
  
drop index if exists model\_description\_name\_unique cascade;  
  
drop table if exists models\_descriptions\_dim cascade;  
  
drop index if exists region\_code\_unique cascade;  
  
drop table if exists regions\_dim cascade;  
  
drop index if exists route\_code\_unique cascade;  
  
drop table if exists routes\_dim cascade;  
  
drop index if exists turbulence\_name cascade;  
  
drop table if exists turbulence\_categories\_dim cascade;  
  
*/\*==============================================================\*/  
/\* Table: aircraft\_types\_dim \*/  
/\*==============================================================\*/*create table aircraft\_types\_dim (  
 type\_id SERIAL not null,  
 name VARCHAR(20) not null,  
 constraint PK\_AIRCRAFT\_TYPES\_DIM primary key (type\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: aircraft\_type\_name\_unique \*/  
/\*==============================================================\*/*create unique index aircraft\_type\_name\_unique on aircraft\_types\_dim (  
 name  
 );  
  
*/\*==============================================================\*/  
/\* Table: aircrafts\_dim \*/  
/\*==============================================================\*/*create table aircrafts\_dim (  
 aircraft\_id SERIAL not null,  
 registration VARCHAR(12) not null,  
 model\_id INT4 null,  
 model\_text VARCHAR(100) null,  
 country\_id INT4 null,  
 production\_date\_id INT4 null,  
 owner\_id INT4 null,  
 constraint PK\_AIRCRAFTS\_DIM primary key (aircraft\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: aircraft\_registration\_unique \*/  
/\*==============================================================\*/*create unique index aircraft\_registration\_unique on aircrafts\_dim (  
 registration  
 );  
  
*/\*==============================================================\*/  
/\* Table: airlines\_dim \*/  
/\*==============================================================\*/*create table airlines\_dim (  
 airline\_id SERIAL not null,  
 name VARCHAR(100) null,  
 alias VARCHAR(64) null,  
 code VARCHAR(8) null,  
 callsign VARCHAR(50) null,  
 country\_id INT4 null,  
 active BOOL null,  
 constraint PK\_AIRLINES\_DIM primary key (airline\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: airline\_code\_unique \*/  
/\*==============================================================\*/*create unique index airline\_code\_unique on airlines\_dim (  
 code  
 );  
  
*/\*==============================================================\*/  
/\* Table: airports\_dim \*/  
/\*==============================================================\*/*create table airports\_dim (  
 airport\_id SERIAL not null,  
 code VARCHAR(8) null,  
 type\_id INT4 null,  
 name VARCHAR(200) null,  
 latitude FLOAT8 null,  
 longitude FLOAT8 null,  
 elevation INT4 null,  
 region\_id INT4 null,  
 gps\_code VARCHAR(8) null,  
 constraint PK\_AIRPORTS\_DIM primary key (airport\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: airport\_code\_unique \*/  
/\*==============================================================\*/*create unique index airport\_code\_unique on airports\_dim (  
 code  
 );  
  
*/\*==============================================================\*/  
/\* Table: airports\_types\_dim \*/  
/\*==============================================================\*/*create table airports\_types\_dim (  
 type\_id SERIAL not null,  
 name VARCHAR(16) not null,  
 constraint PK\_AIRPORTS\_TYPES\_DIM primary key (type\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: airport\_type\_name\_unique \*/  
/\*==============================================================\*/*create unique index airport\_type\_name\_unique on airports\_types\_dim (  
 name  
 );  
  
*/\*==============================================================\*/  
/\* Table: continents\_dim \*/  
/\*==============================================================\*/*create table continents\_dim (  
 continent\_id SERIAL not null,  
 code VARCHAR(2) not null,  
 name VARCHAR(20) not null,  
 constraint PK\_CONTINENTS\_DIM primary key (continent\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: continent\_code\_unique \*/  
/\*==============================================================\*/*create unique index continent\_code\_unique on continents\_dim (  
 code  
 );  
  
*/\*==============================================================\*/  
/\* Table: countries\_dim \*/  
/\*==============================================================\*/*create table countries\_dim (  
 country\_id SERIAL not null,  
 continent\_id INT4 not null,  
 code VARCHAR(2) not null,  
 name VARCHAR(64) not null,  
 constraint PK\_COUNTRIES\_DIM primary key (country\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: country\_code\_unique \*/  
/\*==============================================================\*/*create unique index country\_code\_unique on countries\_dim (  
 code  
 );  
  
*/\*==============================================================\*/  
/\* Table: dates\_dim \*/  
/\*==============================================================\*/*create table dates\_dim (  
 date\_id SERIAL not null,  
 the\_date DATE not null,  
 weekday INT2 not null,  
 month INT4 not null,  
 year INT2 not null,  
 quarter INT2 not null,  
 day\_of\_year INT2 not null,  
 weekend BOOL not null,  
 week\_of\_year INT2 not null,  
 constraint PK\_DATES\_DIM primary key (date\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: time\_date\_unique \*/  
/\*==============================================================\*/*create unique index time\_date\_unique on dates\_dim (  
 the\_date  
 );  
  
*/\*==============================================================\*/  
/\* Table: designators\_dim \*/  
/\*==============================================================\*/*create table designators\_dim (  
 model\_id SERIAL not null,  
 name VARCHAR(255) null,  
 description\_id INT4 null,  
 turbulence\_category\_id INT4 null,  
 designator VARCHAR(4) null,  
 manufacturer\_id INT4 null,  
 type\_id INT4 null,  
 engine\_count INT2 null,  
 engine\_type\_id INT4 null,  
 constraint PK\_DESIGNATORS\_DIM primary key (model\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: designator\_unique \*/  
/\*==============================================================\*/*create unique index designator\_unique on designators\_dim (  
 designator,  
 name  
 );  
  
*/\*==============================================================\*/  
/\* Table: engine\_types\_dim \*/  
/\*==============================================================\*/*create table engine\_types\_dim (  
 engine\_type\_id SERIAL not null,  
 name VARCHAR(30) not null,  
 constraint PK\_ENGINE\_TYPES\_DIM primary key (engine\_type\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: engine\_type\_name\_unique \*/  
/\*==============================================================\*/*create unique index engine\_type\_name\_unique on engine\_types\_dim (  
 name  
 );  
  
*/\*==============================================================\*/  
/\* Table: flights\_fact \*/  
/\*==============================================================\*/*create table flights\_fact (  
 flight\_id INT8 not null,  
 route\_id INT4 null,  
 aircraft\_id INT4 null,  
 airline\_id INT4 null,  
 airport\_origin\_id INT4 null,  
 airport\_destination\_id INT4 null,  
 scheduled\_departure\_date\_id INT4 null,  
 scheduled\_departure\_time TIME null,  
 scheduled\_arrival\_date\_id INT4 null,  
 scheduled\_arrival\_time TIME null,  
 real\_departure\_date\_id INT4 null,  
 real\_departure\_time TIME null,  
 real\_arrival\_date\_id INT4 null,  
 real\_arrival\_time TIME null,  
 constraint PK\_FLIGHTS\_FACT primary key (flight\_id)  
);  
  
*/\*==============================================================\*/  
/\* Table: manufacturers\_dim \*/  
/\*==============================================================\*/*create table manufacturers\_dim (  
 manufacturer\_id SERIAL not null,  
 name VARCHAR(64) not null,  
 constraint PK\_MANUFACTURERS\_DIM primary key (manufacturer\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: manufacturer\_name\_unique \*/  
/\*==============================================================\*/*create unique index manufacturer\_name\_unique on manufacturers\_dim (  
 name  
 );  
  
*/\*==============================================================\*/  
/\* Table: models\_descriptions\_dim \*/  
/\*==============================================================\*/*create table models\_descriptions\_dim (  
 description\_id SERIAL not null,  
 name VARCHAR(3) not null,  
 constraint PK\_MODELS\_DESCRIPTIONS\_DIM primary key (description\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: model\_description\_name\_unique \*/  
/\*==============================================================\*/*create unique index model\_description\_name\_unique on models\_descriptions\_dim (  
 name  
 );  
  
*/\*==============================================================\*/  
/\* Table: regions\_dim \*/  
/\*==============================================================\*/*create table regions\_dim (  
 region\_id SERIAL not null,  
 country\_id INT4 not null,  
 code VARCHAR(8) not null,  
 name VARCHAR(100) not null,  
 constraint PK\_REGIONS\_DIM primary key (region\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: region\_code\_unique \*/  
/\*==============================================================\*/*create unique index region\_code\_unique on regions\_dim (  
 code  
 );  
  
*/\*==============================================================\*/  
/\* Table: routes\_dim \*/  
/\*==============================================================\*/*create table routes\_dim (  
 route\_id SERIAL not null,  
 code VARCHAR(12) not null,  
 constraint PK\_ROUTES\_DIM primary key (route\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: route\_code\_unique \*/  
/\*==============================================================\*/*create unique index route\_code\_unique on routes\_dim (  
 code  
 );  
  
*/\*==============================================================\*/  
/\* Table: turbulence\_categories\_dim \*/  
/\*==============================================================\*/*create table turbulence\_categories\_dim (  
 category\_id SERIAL not null,  
 name VARCHAR(3) not null,  
 constraint PK\_TURBULENCE\_CATEGORIES\_DIM primary key (category\_id)  
);  
  
*/\*==============================================================\*/  
/\* Index: turbulence\_name \*/  
/\*==============================================================\*/*create unique index turbulence\_name on turbulence\_categories\_dim (  
 name  
 );  
  
alter table aircrafts\_dim  
 add constraint FK\_AIRCRAFT\_REFERENCE\_DATES\_DI foreign key (production\_date\_id)  
 references dates\_dim (date\_id)  
 on delete restrict on update cascade;  
  
alter table aircrafts\_dim  
 add constraint FK\_AIRCRAFT\_REFERENCE\_COUNTRIE foreign key (country\_id)  
 references countries\_dim (country\_id)  
 on delete restrict on update restrict;  
  
alter table aircrafts\_dim  
 add constraint FK\_AIRCRAFT\_REFERENCE\_DESIGNAT foreign key (model\_id)  
 references designators\_dim (model\_id)  
 on delete restrict on update restrict;  
  
alter table aircrafts\_dim  
 add constraint FK\_AIRCRAFT\_REFERENCE\_AIRLINES foreign key (owner\_id)  
 references airlines\_dim (airline\_id)  
 on delete restrict on update restrict;  
  
alter table airlines\_dim  
 add constraint FK\_AIRLINES\_REFERENCE\_COUNTRIE foreign key (country\_id)  
 references countries\_dim (country\_id)  
 on delete restrict on update restrict;  
  
alter table airports\_dim  
 add constraint FK\_AIRPORTS\_REFERENCE\_REGIONS\_ foreign key (region\_id)  
 references regions\_dim (region\_id)  
 on delete restrict on update restrict;  
  
alter table airports\_dim  
 add constraint FK\_AIRPORTS\_REFERENCE\_AIRPORTS foreign key (type\_id)  
 references airports\_types\_dim (type\_id)  
 on delete restrict on update restrict;  
  
alter table countries\_dim  
 add constraint FK\_COUNTRIE\_REFERENCE\_CONTINEN foreign key (continent\_id)  
 references continents\_dim (continent\_id)  
 on delete restrict on update restrict;  
  
alter table designators\_dim  
 add constraint FK\_DESIGNAT\_REFERENCE\_ENGINE\_T foreign key (engine\_type\_id)  
 references engine\_types\_dim (engine\_type\_id)  
 on delete restrict on update restrict;  
  
alter table designators\_dim  
 add constraint FK\_DESIGNAT\_REFERENCE\_MODELS\_D foreign key (description\_id)  
 references models\_descriptions\_dim (description\_id)  
 on delete restrict on update restrict;  
  
alter table designators\_dim  
 add constraint FK\_DESIGNAT\_REFERENCE\_AIRCRAFT foreign key (type\_id)  
 references aircraft\_types\_dim (type\_id)  
 on delete restrict on update restrict;  
  
alter table designators\_dim  
 add constraint FK\_DESIGNAT\_REFERENCE\_TURBULEN foreign key (turbulence\_category\_id)  
 references turbulence\_categories\_dim (category\_id)  
 on delete restrict on update restrict;  
  
alter table designators\_dim  
 add constraint FK\_DESIGNAT\_REFERENCE\_MANUFACT foreign key (manufacturer\_id)  
 references manufacturers\_dim (manufacturer\_id)  
 on delete restrict on update restrict;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_REFERENCE\_ROUTES\_D foreign key (route\_id)  
 references routes\_dim (route\_id)  
 on delete restrict on update restrict;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_REFERENCE\_AIRCRAFT foreign key (aircraft\_id)  
 references aircrafts\_dim (aircraft\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_REFERENCE\_AIRLINES foreign key (airline\_id)  
 references airlines\_dim (airline\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_AIRPORT\_D\_AIRPORTS foreign key (airport\_destination\_id)  
 references airports\_dim (airport\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_FLIGHT\_AI\_AIRPORTS foreign key (airport\_origin\_id)  
 references airports\_dim (airport\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_FLIGHT\_RE\_ARR\_t foreign key (real\_arrival\_date\_id)  
 references dates\_dim (date\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_FLIGHT\_RE\_DATES\_DI foreign key (real\_departure\_date\_id)  
 references dates\_dim (date\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_FLIGHT\_SC\_ARR foreign key (scheduled\_arrival\_date\_id)  
 references dates\_dim (date\_id)  
 on delete restrict on update cascade;  
  
alter table flights\_fact  
 add constraint FK\_FLIGHTS\_\_FLIGHT\_SC\_DEP foreign key (scheduled\_departure\_date\_id)  
 references dates\_dim (date\_id)  
 on delete restrict on update cascade;  
  
alter table regions\_dim  
 add constraint FK\_REGIONS\_\_REFERENCE\_COUNTRIE foreign key (country\_id)  
 references countries\_dim (country\_id)  
 on delete restrict on update restrict;

# Додаток Г (ETL скрипти)

## Python-скрипт для завантаження даних у stage зону

def load\_to\_stage():  
 conn = psycopg2.connect(host=DB\_HOST, database=DB\_NAME, user=DB\_USER, password=DB\_PASSWORD)  
 conn.autocommit = True  
 cursor = conn.cursor()  
  
 cursor.execute('TRUNCATE stage.aircrafts;')  
 cursor.execute('TRUNCATE stage.airlines;')  
 cursor.execute('TRUNCATE stage.airports;')  
 cursor.execute('TRUNCATE stage.continents;')  
 cursor.execute('TRUNCATE stage.countries;')  
 cursor.execute('TRUNCATE stage.designators;')  
 cursor.execute('TRUNCATE stage.flights;')  
 cursor.execute('TRUNCATE stage.regions;')  
 cursor.execute('TRUNCATE stage.dates;')  
  
 cursor.execute(  
 f"COPY stage.aircrafts (registration, model\_code, model\_text, country\_code, production\_date, owner\_icao) FROM '{DATA\_DIR.joinpath('aircrafts\_filtered.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.airlines (airline\_id, name, alias, code, callsign, country, active) FROM '{DATA\_DIR.joinpath('airlines.csv')}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.airports (code, type, name, latitude, longitude, elevation, region, gps\_code) FROM '{DATA\_DIR.joinpath('airports.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.continents (code, name) FROM '{DATA\_DIR.joinpath('continents.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.countries (code, name, continent) FROM '{DATA\_DIR.joinpath('countries.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.designators (name, description, turbulence\_category, designator, manufacturer, type, engine\_count, engine\_type) FROM '{DATA\_DIR.joinpath('designators.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.flights (flight\_id, route\_number, aircraft\_registration, airline\_icao, airport\_origin, airport\_destination, scheduled\_departure, scheduled\_arrival, real\_departure, real\_arrival) FROM '{DATA\_DIR.joinpath('flights.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.regions (code, name, country) FROM '{DATA\_DIR.joinpath('regions.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.execute(  
 f"COPY stage.dates (the\_date, weekday, month, year, quarter, day\_of\_year, weekend, week\_of\_year) FROM '{DATA\_DIR.joinpath('dates.csv').absolute()}' DELIMITER ',' CSV HEADER;")  
  
 cursor.close()  
 conn.close()

## SQL-скрипт для завантаження даних зі stage зони до основного сховища даних (окрім вимірів літаків та польотів)

*-- Designators*INSERT INTO public.manufacturers\_dim (name)  
SELECT DISTINCT manufacturer  
FROM stage.designators  
WHERE manufacturer IS NOT NULL  
ON CONFLICT (name) DO NOTHING;  
  
INSERT INTO public.turbulence\_categories\_dim (name)  
SELECT DISTINCT turbulence\_category  
FROM stage.designators  
WHERE turbulence\_category IS NOT NULL  
ON CONFLICT (name) DO NOTHING;  
  
INSERT INTO public.aircraft\_types\_dim (name)  
SELECT DISTINCT type  
FROM stage.designators  
WHERE type IS NOT NULL  
ON CONFLICT (name) DO NOTHING;  
  
INSERT INTO public.models\_descriptions\_dim (name)  
SELECT DISTINCT description  
FROM stage.designators  
WHERE description IS NOT NULL  
ON CONFLICT (name) DO NOTHING;  
  
INSERT INTO public.engine\_types\_dim (name)  
SELECT DISTINCT engine\_type  
FROM stage.designators  
WHERE designators.engine\_type IS NOT NULL  
ON CONFLICT (name) DO NOTHING;  
  
  
*-- Geo zones*INSERT INTO public.continents\_dim (code, name)  
SELECT DISTINCT ON (stage.continents.code) code, name  
FROM stage.continents  
ORDER BY code, stage.continents.continent\_id DESC  
ON CONFLICT (code) DO UPDATE SET name = excluded.name;  
  
INSERT INTO public.countries\_dim (code, continent\_id, name)  
SELECT DISTINCT ON (stage.countries.code) stage.countries.code,  
 public.continents\_dim.continent\_id,  
 stage.countries.name  
FROM stage.countries  
 INNER JOIN public.continents\_dim ON stage.countries.continent = public.continents\_dim.code  
ORDER BY code, stage.countries.country\_id DESC  
ON CONFLICT (code) DO UPDATE SET continent\_id = excluded.continent\_id,  
 name = excluded.name;  
  
INSERT INTO public.regions\_dim (code, country\_id, name)  
SELECT DISTINCT ON (stage.regions.code) stage.regions.code,  
 public.countries\_dim.country\_id,  
 stage.regions.name  
FROM stage.regions  
 INNER JOIN public.countries\_dim ON stage.regions.country = public.countries\_dim.code  
ORDER BY code, stage.regions.region\_id DESC  
ON CONFLICT (code) DO UPDATE SET country\_id = excluded.country\_id,  
 name = excluded.name;  
  
  
*-- Airports*INSERT INTO public.airports\_types\_dim (name)  
SELECT DISTINCT stage.airports.type  
FROM stage.airports  
ON CONFLICT (name) DO NOTHING;  
  
INSERT INTO public.airports\_dim (code, type\_id, name, latitude, longitude, elevation, region\_id, gps\_code)  
SELECT DISTINCT ON (stage.airports.code) stage.airports.code,  
 public.airports\_types\_dim.type\_id,  
 stage.airports.name,  
 stage.airports.latitude,  
 stage.airports.longitude,  
 stage.airports.elevation,  
 public.regions\_dim.region\_id,  
 stage.airports.gps\_code  
FROM stage.airports  
 LEFT JOIN public.airports\_types\_dim ON stage.airports.type = public.airports\_types\_dim.name  
 LEFT JOIN public.regions\_dim ON stage.airports.region = public.regions\_dim.code  
ORDER BY stage.airports.code, stage.airports.airport\_id DESC  
ON CONFLICT (code) DO UPDATE SET type\_id = excluded.type\_id,  
 name = excluded.name,  
 latitude = excluded.latitude,  
 longitude = excluded.longitude,  
 elevation = excluded.elevation,  
 region\_id = excluded.region\_id,  
 gps\_code = excluded.gps\_code;  
  
  
*-- Airlines*INSERT INTO public.airlines\_dim (code, airline\_id, name, alias, callsign, country\_id, active)  
SELECT DISTINCT ON (stage.airlines.code) stage.airlines.code,  
 stage.airlines.airline\_id,  
 stage.airlines.name,  
 stage.airlines.alias,  
 stage.airlines.callsign,  
 public.countries\_dim.country\_id,  
 stage.airlines.active  
FROM stage.airlines  
 LEFT JOIN public.countries\_dim ON stage.airlines.country = public.countries\_dim.name  
ORDER BY stage.airlines.code, stage.airlines.airline\_pk\_ser DESC  
ON CONFLICT (airline\_id) DO UPDATE SET alias = excluded.alias,  
 code = excluded.code,  
 callsign = excluded.callsign,  
 country\_id = excluded.country\_id,  
 active = excluded.active;  
  
  
*-- Routes*INSERT INTO public.routes\_dim (code)  
SELECT DISTINCT stage.flights.route\_number  
FROM stage.flights  
ON CONFLICT (code) DO NOTHING;  
  
*-- Dates*INSERT INTO public.dates\_dim (the\_date, weekday, month, year, quarter, day\_of\_year, weekend, week\_of\_year)  
SELECT DISTINCT ON (stage.dates.the\_date) stage.dates.the\_date,  
 stage.dates.weekday,  
 stage.dates.month,  
 stage.dates.year,  
 stage.dates.quarter,  
 stage.dates.day\_of\_year,  
 stage.dates.weekend,  
 stage.dates.week\_of\_year  
FROM stage.dates  
ORDER BY stage.dates.the\_date, stage.dates.date\_id DESC  
ON CONFLICT (the\_date) DO UPDATE SET weekend = excluded.weekend;  
  
  
*-- Designators*INSERT INTO designators\_dim (name, designator, description\_id, turbulence\_category\_id, manufacturer\_id, type\_id,  
 engine\_count, engine\_type\_id)  
SELECT DISTINCT ON (stage.designators.name, stage.designators.designator) stage.designators.name,  
 stage.designators.designator,  
 public.models\_descriptions\_dim.description\_id,  
 public.turbulence\_categories\_dim.category\_id,  
 public.manufacturers\_dim.manufacturer\_id,  
 public.aircraft\_types\_dim.type\_id,  
 stage.designators.engine\_count,  
 public.engine\_types\_dim.engine\_type\_id  
FROM stage.designators  
 LEFT JOIN public.models\_descriptions\_dim ON public.models\_descriptions\_dim.name = stage.designators.description  
 LEFT JOIN public.turbulence\_categories\_dim  
 ON public.turbulence\_categories\_dim.name = stage.designators.turbulence\_category  
 LEFT JOIN public.manufacturers\_dim ON public.manufacturers\_dim.name = stage.designators.manufacturer  
 LEFT JOIN public.aircraft\_types\_dim ON public.aircraft\_types\_dim.name = stage.designators.type  
 LEFT JOIN public.engine\_types\_dim ON public.engine\_types\_dim.name = stage.designators.engine\_type  
ORDER BY stage.designators.name, stage.designators.designator, stage.designators.model\_id DESC  
ON CONFLICT (name, designator) DO UPDATE SET description\_id = excluded.description\_id,  
 turbulence\_category\_id = excluded.turbulence\_category\_id,  
 manufacturer\_id = excluded.manufacturer\_id,  
 type\_id = excluded.type\_id,  
 engine\_count = excluded.engine\_count,  
 engine\_type\_id = excluded.engine\_type\_id;  
  
DROP TABLE IF EXISTS temp\_aircrafts;

## Python-скрипт для створення нової таблиці з літаками та додаванням до неї стовпця назви моделі з таблиці моделей

def link\_models(conn: str):  
 aircrafts = pd.read\_sql\_table('aircrafts', conn, schema='stage', index\_col='aircraft\_id')  
 designators = pd.read\_sql\_table('designators\_dim', conn, index\_col='model\_id')  
 results = []  
 progress\_bar = tqdm.tqdm(list(aircrafts.iterrows()))  
 for row in progress\_bar:  
 \_, model\_code, model\_text, \*\_ = row[1]  
 progress\_bar.set\_description(f'{model\_code} | {model\_text}')  
 progress\_bar.refresh()  
 variants = designators.query(f'designator == "{model\_code}"').name  
 result = None  
 if model\_text and not variants.empty:  
 ratios = np.array([SequenceMatcher(None, model\_text, variant).ratio() for variant in variants])  
 result = variants.iloc[ratios.argmax()]  
 results.append(result)  
 aircrafts['linked\_model\_text'] = results  
 return aircrafts

## SQL-скрипт для завантаження даних зі стейдж зони до основного сховища даних (для виміру літаків та польотів)

*-- Aircrafts*SELECT *\**FROM temp\_aircrafts;  
INSERT INTO public.aircrafts\_dim (registration, model\_id, model\_text, country\_id, production\_date\_id, owner\_id)  
SELECT DISTINCT ON (temp\_aircrafts.registration) temp\_aircrafts.registration,  
 public.designators\_dim.model\_id,  
 temp\_aircrafts.model\_text,  
 public.countries\_dim.country\_id,  
 public.dates\_dim.date\_id,  
 public.airlines\_dim.airline\_id  
FROM temp\_aircrafts  
 LEFT JOIN public.designators\_dim ON temp\_aircrafts.model\_code = public.designators\_dim.designator AND  
 temp\_aircrafts.linked\_model\_text = public.designators\_dim.name  
 LEFT JOIN public.countries\_dim ON temp\_aircrafts.country\_code = public.countries\_dim.code  
 LEFT JOIN public.dates\_dim ON temp\_aircrafts.production\_date = public.dates\_dim.the\_date  
 LEFT JOIN public.airlines\_dim ON temp\_aircrafts.owner\_icao = public.airlines\_dim.code  
ORDER BY temp\_aircrafts.registration, temp\_aircrafts.aircraft\_id DESC  
ON CONFLICT (registration) DO UPDATE SET model\_id = excluded.model\_id,  
 model\_text = excluded.model\_text,  
 country\_id = excluded.country\_id,  
 production\_date\_id = excluded.production\_date\_id,  
 owner\_id = excluded.owner\_id;  
DROP TABLE IF EXISTS temp\_aircrafts;  
  
INSERT INTO public.flights\_fact (flight\_id, route\_id, aircraft\_id, airline\_id, airport\_origin\_id,  
 airport\_destination\_id, scheduled\_departure\_date\_id, scheduled\_departure\_time,  
 scheduled\_arrival\_date\_id, scheduled\_arrival\_time, real\_departure\_date\_id,  
 real\_departure\_time, real\_arrival\_date\_id, real\_arrival\_time)  
SELECT flight\_id,  
 public.routes\_dim.route\_id,  
 public.aircrafts\_dim.aircraft\_id,  
 public.airlines\_dim.airline\_id,  
 ap1.airport\_id,  
 ap2.airport\_id,  
 d1.date\_id,  
 scheduled\_departure::TIME,  
 d2.date\_id,  
 scheduled\_arrival::TIME,  
 d3.date\_id,  
 real\_departure::TIME,  
 d4.date\_id,  
 real\_arrival::TIME  
FROM (SELECT DISTINCT ON (stage.flights.flight\_id) stage.flights.flight\_id,  
 stage.flights.route\_number,  
 stage.flights.aircraft\_registration,  
 stage.flights.airline\_icao,  
 stage.flights.airport\_origin,  
 stage.flights.airport\_destination,  
 *TO\_TIMESTAMP*(stage.flights.scheduled\_departure) AS scheduled\_departure,  
 *TO\_TIMESTAMP*(stage.flights.scheduled\_arrival) AS scheduled\_arrival,  
 *TO\_TIMESTAMP*(stage.flights.real\_departure) AS real\_departure,  
 *TO\_TIMESTAMP*(stage.flights.real\_arrival) AS real\_arrival  
 FROM stage.flights  
 ORDER BY flight\_id DESC) AS flights\_subq  
 LEFT JOIN public.routes\_dim ON route\_number = public.routes\_dim.code  
 LEFT JOIN public.aircrafts\_dim ON aircraft\_registration = public.aircrafts\_dim.registration  
 LEFT JOIN public.airlines\_dim ON airline\_icao = public.airlines\_dim.code  
 LEFT JOIN public.airports\_dim ap1 ON airport\_origin = ap1.code  
 LEFT JOIN public.airports\_dim ap2 ON airport\_destination = ap2.code  
 LEFT JOIN public.dates\_dim d1 ON scheduled\_departure::date = d1.the\_date  
 LEFT JOIN public.dates\_dim d2 ON scheduled\_arrival::date = d2.the\_date  
 LEFT JOIN public.dates\_dim d3 ON real\_departure::date = d3.the\_date  
 LEFT JOIN public.dates\_dim d4 ON real\_arrival::date = d4.the\_date  
ON CONFLICT (flight\_id) DO UPDATE SET route\_id = excluded.route\_id,  
 aircraft\_id = excluded.aircraft\_id,  
 airline\_id = excluded.airline\_id,  
 airport\_origin\_id = excluded.airport\_origin\_id,  
 airport\_destination\_id = excluded.airport\_destination\_id,  
 scheduled\_departure\_date\_id = excluded.scheduled\_departure\_date\_id,  
 scheduled\_departure\_time = excluded.scheduled\_departure\_time,  
 scheduled\_arrival\_date\_id = excluded.scheduled\_arrival\_date\_id,  
 scheduled\_arrival\_time = excluded.scheduled\_arrival\_time,  
 real\_departure\_date\_id = excluded.real\_departure\_date\_id,  
 real\_departure\_time = excluded.real\_departure\_time,  
 real\_arrival\_date\_id = excluded.real\_arrival\_date\_id,  
 real\_arrival\_time = excluded.real\_arrival\_time;

## Python-скрипт для повного завантаження даних

def etl():  
 load\_to\_stage()  
 stage\_to\_fact()